International Journal of Electrical and Electronics Engineers
Vol. No.8 Issue 01, January-June 2016
www.arresearchpublication.com
IJEEE
ISSN 2321 - 2055

MATERIALIZE BEHAVIORAL MODELTHROUGH ACTIVITY DIAGRAM IN TESTING AND OPTIMIZATION

Ami Tiwari[1] , Dr. Amit Sharma[2]

[1] *M.Tech Scholar Department of Electrical Engineering ,*
Vedant College of Engineering & Technology ,Bundi ,Rajasthan, (India)
[2] *Associate Professor Department of Computer Science & Engineering ,*
Vedant College of Engineering & Technology ,Bundi ,Rajasthan, (India)

ABSTRACT

In this research Paper I am focusing on The Unified Modeling Language (UML) is a standard notation used to model user's requirements for object oriented software systems. With the growing adoption of UML by software developers and researchers, studies have begun to investigate how it can be used to conduct testing. Several approaches of UML-based software testing have been proposed where test cases are derived from UML diagrams based on efficient algorithms. However, these approaches still suffer from three major limitations namely; inadequate test coverage criteria, insufficient diagrams coverage's and incompatibility with various UML tools. Therefore, this paper proposes an improved approach for generating test cases from various UML diagrams based on full coverage criteria. To achieve this aim, a robust method for extracting artifacts from the underlying diagrams of the software under test (SUT) was developed, where the artifacts are represented in an intermediate form using a tree and test cases are generated by traversing the contents of the tree. The proposed approach is fully automated and the generated test cases satisfied the criteria defined for the generation process. The novelty of this approach is its systematic rather than ad hoc test case generation from UML diagrams to achieve high test coverage.

Keywords-component: UML, SUT, requirements, software, test case generation

I. INTRODUCTION

In this chapter, the content and rationale of this thesis is introduced with a brief description of every chapter and appendix. Moreover, some basic questions that we think the reader of this thesis can consider are answered .Finally, some typographic conventions that are used throughout this thesis are explained.

Unified Modeling Language (UML) [1] was introduced in the late 90s, its primary application was to support the communication among software engineers. UML provided a standardized way to describe software artifacts. There were only limited and rarely successful attempts to reuse UML artifacts for the implementation phase of a software product. The **UML** is a standard diagramming notation.

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems [OMG01].

With the advent of methodologies like Model Driven Architecture (MDA) [2] and the Model Driven Software Development (MDSD) [3] and platforms like Eclipse Modeling Technology (EMT) [4], a new wave of code generation tools emerged [5,6,7]. The focus of these technologies is on the design of abstract models that avoid implementation details, which can automatically be transformed into code by means of intelligent code generators.

The widespread usage of UML indicates that modeling is one of the key elements in many software development projects. The evolution of MDA and its associated technologies drastically altered software development processes. There is no restriction regarding a specific programming language, instead the focus lies on developing UML models based on the requirements and translating them into an executable software system in general.

II. UML BEHAVIORAL MODELS

Behavioral diagrams depict the behavioral features of a system or business process. Behavioral diagrams include the following diagram types:

Diagram Type	Detail
Activity Diagrams	Activity diagrams model the behaviors of a system, and the way in which these behaviors are related in an overall flow of the system.
Use Case Diagrams	Use Case diagrams capture Use Cases and relationships among Actors and the system; they describes the functional requirements of the system, the manner in which external operators interact at the system boundary, and the response of the system.
Diagram Type	**Detail**
State Machine Diagrams	State Machine diagrams illustrate how an element can move between states, classifying its behavior according to transition triggers and constraining guards.
Timing Diagrams	Timing diagrams define the behavior of different objects within a time-scale, providing a visual representation of objects changing state and interacting over time.
Sequence Diagrams	Sequence diagrams are structured representations of behavior as a series of sequential steps over time. They are used to depict work flow, message passing and how elements in general cooperate over time to achieve a result.
Communication Diagrams	Communication diagrams show the interactions between elements at run-time, visualizing inter-object relationships.
Interaction Overview Diagrams	Interaction Overview diagrams visualize the cooperation between other interaction diagrams (Timing, Sequence, Communication and Interaction Overview diagrams) to illustrate a control flow serving an encompassing purpose.

III. PROBLEM FORMULATION

UML and model based testing

Object-oriented analysis and design has come into existence, it has found widespread acceptance in the industry as well as in academics. The main reason for the popularity of OOAD is as follows:

• Code and design reuse

• Increased productivity

• Ease of testing and maintenance

• Better code and design understandability

UML has helped a lot to visualize/realize the software development process. UML accomplish the visualization of software at early stage of SDLC, which helps in many ways like confidence of both developer and the end user on the system, earlier error detection through proper analysis of design and etc. UML also helps in making the proper documentation of the software and so maintains the consistency in between the specification and design document.

Model-based software testing generally refers to test case design intermediate artifacts between requirement specification and final code. Models preserve the essential information from the requirement, and are the basis for implementation. Therefore, models concisely describe the structural and behavioral aspects, are necessary for implementation of the software. Model based testing can be summarized in one sentence; "it is essentially a technique for automatic generation of test cases from specified software model". The key advantage of this technique is that the test generation can systematically derive all combination of tests associated with the requirements represented in the model to automate both the test design and test execution process.

IV. HYPOTHESIS

Section 3 reviews and discusses the competent programmer hypothesis states that a program written by a competent programmer can be incorrect but it will be slightly different from the correct program. Coupling effect is the relationship between test data which is based on the fact that data that can detect the mutants with simple faults can also detect the more complex faults as well.

- Provides complete, consistent and accurate requirements of the system that need to be tested and,
- Abstracts details to minimize the testing cost.
- Developing a model at the right level of abstraction for effective testing is one of the main challenges for model-based testing
- As the test cases are directly derived from models of the system under test, the effectiveness of the test cases depends on the information available in the model.

V. METHODOLOGY

The test case generation approach consists of four parts such as:

(1) I/O Specification

(2) Designing the document (UML Diagram)

(3) Sub-Optimal Test Case Generation

(4) Test Case Evaluation

5.1 I/O Specifications apple

I/O **Specifications** shows the Input and Output of the projected software in detail. This document is used as primary source of input for test case generation.

5.2 Designing the Document

The activity diagram is used because of its dynamic behavior of modeling. We can visualize, construct, specify and document the dynamic aspects of an object. It is modeled to show the control flow of an operation (or from activity to activity). Activities ultimately result in some action, which is some set of pure computation. An important fact about the collaboration and activity diagrams is that they are most useful for constructing executable systems through forward and reverse engineering.

5.3 Sub-Optimal Test Case Generation

The optimized test cases are generated by applying Genetic Algorithm technique on the input domain. We have considered transition coverage, a test adequacy criteria. The input values are defined in the form of set of sequence of events. An event consists of a name and a list of possible arguments, which when triggered, generate transition from one activity to another. There are lots of input values for an operation. First, all the possible input values are taken in to consideration and then the C-GA is applied on these input domains, so as to minimize the input range. The obtained test cases will be used for further processing. Genetic algorithm is used for generation of better (sub-optimal) test cases. The fitness function is defined to generate the sub-optimal test case. Minimum number of errors detected measure the quality of the test case. The proposed error minimization technique is used to minimize the presence of errors, as we cannot guarantee the complete absence of error but we can minimize the percentage of presence of errors

The above said technique (error minimization) is defined as the difference between the weight age value of expected transition coverage and actual transition coverage.

5.4 Test Case Evaluation

A pass/fail (Boolean variable such as 1 for Pass and 0 for fail) technique is deployed to evaluate the test cases. The input for this step is the optimized test case, obtained from the previous step. The test cases those will only meet all the I/O Specifications are considered as passed test cases, others will be treated as failed test cases.

5.4.1 Genetic Algorithm for our approach

Set of sequence of all events are considered as input domain for the problem. So we have assigned a fitness value to each and every event or transition based on the intended activity to be performed. We have given more weighted value to those events, that involves more branches or decision. Transitions producing simple transitions are given with weight age value 1 and 0 for transitions not producing any transitions. Whereas transitions producing branches or fork and joins are assigned the more weight age value that is 2. Initially, we select randomly a valid set of transitions for the given activity. Then, we generate new solution in the next generation by performing some basic GA operations i.e. selection, crossover and mutation. The best fit test case is selected based on the calculated fitness value. The process is continued until reaching the stopping condition as defined by the user. Any successful test cannot guarantee the absence of error, rather it detects the error. So,

we have deployed an error minimization technique to minimize the percentage of error presence.

VI. CONCLUSION AND FUTURE WORK

A robust approach for UML-based testing was the focus of this research work. It was implemented with Java and fully automated. This has led to the efficient parsing of artefacts from any XMI files of UML diagrams. The proposed approach was validated using project requirements where five different UML diagrams were drawn to model requirements of four different software applications. The diagrams were then used to evaluate the performance of the proposed approach. It performed better that the existing ones in terms of its ability to parse artefacts from XMIs of any UML diagram or model and generate test cases. This is an improvement over existing techniques. With the proposed technique, test cases generation becomes comprehensible since artefacts are parsed based on adequate coverage criteria and not randomly selected criteria. The average accuracy of the generated test cases is 98.00%. As for the future work, we hope to integrate our approach with other model file formats such as .MDL. Also, a technique capable of generating test cases for non-functional requirements such behavior, software usability, security and reliability is worth investigation. Finally, the validation of this technique in industrial setting is considered as a limitation as well.

REFERENCES

[1] Roberto S. Silva Filho, Christof J. Budnik,William M. Hasling, Monica McKenna, Rajesh Subramanyam, “Supporting Concern- Based Regression Testing and Prioritization in a Model-Driven Environment”, 34th Annual IEEE Computer Software and Applications Conference Workshops,2010.

[2] Mr. Rohit N. Devikar, Prof. Manjushree D. Laddha, “Automation of Model-based Regression Testing”, Internatonal Journal of Scientific and Research Publications, Volume 2, Issue 12, December 2012.

[3] A. Srivastava and J. Thiagarajan, "Effectively Prioritizing Tests in Development Environment," in Intl. Symposium on Software Testing and Analysis Roma, Italy: 2002.

[4] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," in Future of Software Engineering: IEEE Computer Society, 2007.

[5] L. C. Briand, Y. Labiche, and S. He, "Automating regression test selection based on UML designs," Inf. Softw. Technol., vol. 51, pp. 16-30, 2009

[6] Y. Chen, R. L. Probert, and D. P. Sims, "Specification-based Regression Test Selection with Risk Analysis," in 2002 Conference of the Centre for Advanced Studies on Collaborative Research Toronto, Ontario, Canada: IBM Press, 2002.

[7] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. EMF -- Eclipse Modeling Framework, Second Edition.Addison-Wesley, 2009. [cited at p. 1]

[8] B. Unhelkar. Verification and Validation For Quality Of UML 2.0 Models. Wiley Interscience, 2005. [cited at p. 9, 14]

[9] P. Chen. The Entity-Relationship Model – toward a Unified View of Data. ACM Transactions on Database Systems (TODS), 1(1):9–36, March 1976.

[10] R. Kimball. The Data Warehouse Toolkit.John Wiley & Sons, 1996. (Last edition: 2nd edition, John Wiley & Sons, 2002).

[11] I. Jacobson, G. Booch, and J. Rumbaugh. The Unified Software Development Process. Object Technology Series. Addison-Wesley, 1999.

[12] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. Object-Oriented Modeling and Design. Prentice Hall, 1992.

[13] G. Booch. Object-Oriented Analysis and Design with Applications. Addison-Wesley, 2 edition, 1994.

[14] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, 1992.

[15] Object Management Group (OMG). Unified Modeling Language (UML) Specification 1.5. Internet: http://www.omg.org/cgi-bin/doc?formal/03-03-01, March 2003.

[16] AshalathaNayak, DebasisSamanta: "Automatic Test Data Synthesis using UML Sequence Diagrams", in Journal of Object Technology, vol. 09, no. 2, March{April 2010, pp. 75 104.

[17] Li Bao-Lin, Li Zhi-shu, Li Qing, Chen Yan Hong ," Test Case automate Generation from UML Sequence diagram and OCL Expression", International Conference on Computational Intelligence and Security 2007, pp 1048-52.

[18] A.V.K. Shanthi, Dr.G.Mohan Kumar, "Automated Test Case From UML Diagram Using Data Mining Approach", CiiT International Journal of Software Engineering and Technology, Vol3.No3, March 2011.

[19] A.V.K. Shanthi, Dr.G.Mohan Kumar, "Automated Test Cases Generation For Object Oriented Software", Indian Journal of Computer Science and Engineering, Vol:2, issue 4,Sep2011.

[20] RANJITA KUMARI SWAIN (2013)"Generation Of Test Cases Using Activity Diagram" International Journal of Computer Science and Informatics, ISSN (PRINT): 2231 –5292, Volume-3, Issue-2, 2013